Three-dimensional Electromagnetic Particle-In-Cell Code using High Performance Fortran on PC Cluster

DongSheng Cai[1], Yaoting Li[1], Ken-ichi Nishikawa[2], Chiejie Xiao[1], and Xiaoyan Yan[1]

[1] Institute of Information Sciences and Electronics,
The University of Tsukuba, Ibaraki 305-8573, Japan
{cai, ytli, cjxiao, yxy }@is.tsukuba.ac.jp
[2] Department of Physics and Astronomy, Rutgers University,
136 Frelinghuysen Road, Piscataway, New Jersey 08854-8019, USA,
kenichi@physics.rutgers.edu

Abstract. A three-dimensional full electromagnetic particle-in-cell (PIC) code, TRISTAN (Tridimensional Stanford) code, has been parallelized using High Performance Fortran (HPF) as a RPM (Real Parallel Machine). In the simulation, the simulation domains are decomposed in one-dimension, and both the particle and field data located in each domain that we call the sub-domain are distributed on each processors. Both the particle and field data on a sub-domain is needed by the neighbor sub-domains and thus communications between the sub-domains are inevitable. Our simulation results using HPF exhibits the promising applicability of the HPF communications to a large scale scientific computing such as 3D particle simulations.

1 Introduction

This paper reports on parallelization of Tridimensional Stanford (TRISTAN) code [1] that is a three-dimensional electromagnetic full particle code developed at Stanford University on a two-way PentiumPro PC cluster that consists of 10 distributed SMPs using High Performance Fortran.

In our parallel program, the simulation domain is decomposed into the sub-domains as shown in Fig. 1. The Particle-In-Cell (PIC) computation in TRISTAN to be performed on a certain sub-domain or on a certain processor where the sub-domain is distributed will typically require the data from their neighbor processors to proceed the whole PIC simulations. Here we distribute the field arrays and the particles over processors as indicated in Fig. 1. Thus the data must be transferred between processors in each time step so as to allow PIC simulation to proceed in time. These inter-processor communications in each time step need to be programmed in HPF constructs.

The amount of inter-processor communications needed for a parallel program basically depends on the algorithms and the scales of the physical problem sizes

adopted in the simulations. In PIC simulations, they are the way decompose the simulation domains, the sizes of the sub-domain boundaries, and the number of the particles in a cell, respectively.

The pgHPF [2] compiler of Portland Group Inc. aims to realize the standard High Performance Fortran specification and can be installed on a number of parallel machines. Executables produced by the pgHPF compilers are unconstrained, and can be executed on any compatible IA-32 processor-based system regardless of whether the pgHPF compilers are installed on that system or not. From the HPF programmer's point of view, the differences between versions of the pgHPF runtime library have little effect on program developments.

In parallel programming models, usually, the SPMD models using MPI (Message Passing Interface) or PVM (Parallel Virtual Machine) are one of the most poplar model. The biggest HPF advantage is its programming styles. Once the simulation domains are decomposed and the data are distributed to each subdomains or over processors using simple HPF compiler directives, other HPF programming styles are very similar to those in usual Fortrans. Of course, the biggest problems here is the performance issues comparing with those using MPI or PVM.

Actually, pgHPF is based on a RPM (PGI Proprietary Communications - Real Parallel Machine) protocol. This transport mechanism was developed by PGI to model the behavior of PVM among a homogeneous group of hosts on a network. It offers both greater programming efficiency and performance than PVM with fewer requirements. In this paper, to archive a similar high parallel performance comparing with that of MPI or PVM using HPF in a full electromagnetic PIC simulation, some careful optimizations of inter-processor communications are proposed.

Our code is same as the TRISTAN code except the parallelization part, which utilizes charge-conserving formulas and radiating boundary conditions [1]. It is written in HPF so that that the code can be run on any parallel computers with the HPF compilers.

The parallelization part of our HPF TRISTAN code is similar to Liewer et al. [3] and Decyk [4]. We separate the communication parts from computation parts, and use both the the particle manager and the field manager to localize the inter-processor communications [4].

The basic controlling equations of the plasmas are:

$$m_i \frac{d\boldsymbol{v}_i}{dt} = q_i \cdot (\boldsymbol{E} + \boldsymbol{v}_i \times \boldsymbol{B}) \tag{1}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} = - \bigtriangledown \times \boldsymbol{E} \tag{2}$$

$$\frac{\partial \boldsymbol{E}}{\partial t} = c^2 \bigtriangledown \times \boldsymbol{B} - \frac{1}{\varepsilon_0} \boldsymbol{J} \tag{3}$$

$$\boldsymbol{J} = n \sum (q_i \boldsymbol{v}_i) \tag{4}$$

The coordinate and one-dimensional domain decomposition using in the simulation domain is shown in Fig. 1. For parallel benchmarking purposes, we perform the real simulations of solar wind-magnetosphere interactions using the code. For the simulation of solar wind-magnetosphere interactions, the following boundary conditions were used for the particles [1]: (1) Fresh particles representing the incoming solar wind (unmagnetized in our test run) are continuously injected across the yz plane at $x = x_{min}$ with a thermal velocity plus a bulk velocity in the $+x$ direction; (2) thermal solar particle flux is also injected across the sides of our rectangular computation domain; (3) escaping particles are arrested in a buffer zone, redistributed there more uniformly by making the zone conducting in order to simulate their escape to infinity, and finally written off. We use a simple model for the ionosphere where both electrons and ions are reflected by the Earth dipole magnetic field. The effects of the Earth rotation are not included. Since the solar-winds and the Earth dipole magnetic field are included, the load-imbalance due to this asymmetry are expected in this HPF TRISTAN code.

2 Arrays in Original TRISTAN code

The motivation of TRISTAN, a fully three-dimensional (3D) electromagnetic (EM) particle-in-cell(PIC) code written by Oscar Buneman and other collaborators in Stanford University, is to develop a general particle-in-cell code for space plasma simulations [1]. Here we only discuss the data structure and the data distribution over processors on the HPF TRISTAN code, and do not discuss the details of the numerical algorithms and the plasma physics in TRISTAN in the present report. For the physics of the PIC code, please refer to, for examples, [5] and [6].

The data structure of TRISTAN code consists of two primitive data types. The first one is the particle data as follows: $x(mp), y(mp), z(mp), u(mp), v(mp), w(mp)$, where mp= total number of particles, the positions and velocities of ions and electrons are recorded at $x(1 : mh), y(1 : mh), z(1 : mh), u(1 : mh), v(1 : mh), w(1 : mh)$, and $x(mh+1 : mp), y(mh+1 : mp), z(mh+1 : mp), u(mh+1 : mp), v(mh+1 : mp), w(mh+1 : mp)$, respectively, where $mh = mp/2$. The second one is the grided field data expressed as the triple-indexed arrays of EM (ElectroMagnetic) fields as follows:

$$ex(i, j, k), ey(i, j, k), ez(i, j, k),$$

and

$$bx(i, j, k), by(i, j, k), bz(i, j, k).$$

The original TRISTAN code uses "COMMON" block clause to save and transfer fields data between subroutines in the **MOVER**(push particles) and **DEPOSIT** (deposit current data to the field grids) subroutine calls. Meanwhile in the subroutines that processes the surfaces and edges of the grid data, the filed data are transferred by dummy arrays in the original code. In both of these subroutines,

the field arrays are treated as single-indexed. On the other hand, triple-indexed field arrays are employed in the field solver subroutines. In the code, single-indexed arrays are converted automatically to the triple-indexed arrays when they passed over two subroutines.

Converting a serial Fortran program to a HPF program, we have to stress two points that are very important for rewriting TRISTAN in HPF: **1** the "COMMON" statement is restricted as suggested by pgHPF user guide and there they indicated 'We strongly recommended that programmers writing new F90 code use features like "MODULE" ... to avoid the use of "COMMON"...'[7][8], in case of data overlapping, and substituted it by "MODULE" block; and **2** to control the communications, all the arrays are treated as fixed indexes throughout the whole program. We control the communication parts using both the field and particle managers [4].

3 Field Data Domain Decompositions

The field data are decomposed over sub-domains of that number is equal to the number of the processors as indicated in Fig. 1. In processing the current deposition that is so-called the scatter part of the computations, to avoid large transients or variations of currents TRISTAN uses a 'smoother' that has 27 different weights, smoothing the current deposition. In **DEPOSIT** subroutine the smoothing is performed as follows:

ey(i+smx+1,j+smy,k+smz+1, Np)=ey(i+smx+1,j+smy,k+smz+1, Np)+...,

where $smx = -1 : 1, smy = -1 : 1, smz = -1 : 1$. Therefore, the current deposition of one particle will be related to three grids in each dimension, where one of them are at the backward grid and two of them at the forward grids in each dimension.

In the "MODULE" block, the field arrays are written in HPF directives as follows:

$$\mathbf{REAL}, \mathbf{DIMENSION}(nx, j, k, Np) :: ex, ey, ez$$

$$\mathbf{REAL}, \mathbf{DIMENSION}(nx, j, k, Np) :: bx, by, bz$$

where Np=the number of processor, $nx = i/Np + 3$ (here assuming i/Np is not necessarily equal to be integer exactly) keeping one guard cell in the left (backward) and two in right (forward) side of the sub-domains in the domain-decomposition direction (i. e., in the solar-magnetotail direction). Here the indices i, j and k corresponds to the numbers of field grids in x, y and z directions, respectively. Using the HPF directive "DISTRIBUTE", we, respectively, map the sub-domains to each processor on a distributed memory parallel computer:

DISTRIBUTE(*,*,*,BLOCK) ONTO Np :: ex,ey,ez

DISTRIBUTE(*,*,*,BLOCK) ONTO Np :: bx,by,bz

In order to separate the communication parts from the computation parts, each sub-domain keeps extra cells, the so-called guard or ghost cells, that store the field data information in the first and the last two grids of that sub-domain in the decomposition direction. Figure 2 illustrates this concept of the data mapping over the sub-domains or processors. Here the communications are required after updating the field data every time step. In the field manager [4], the data send to the neighbor processors are packed in the working arrays: $Cex(1, j, k, Np)$, $Cey(1, j, k, Np)$, and $Cez(1, j, k, Np)$, before they are send to the neighbor sub-domains. Thus the field data communications are performed by the HPF $CSHIFT$ construct after the data are packed in the working arrays. The followings are the related parts of the HPF programs in the field manager[4]:

```
Cex(1,:,:,:)=ex(2,:,:,:)
Cex=CSHIFT(Cex,+1,4)
ex(nx-1,:,:,:)=Cex(1,:,:,:)
...
```

4 Particle Data Domain Decompositions

The particle data can be written in HPF directives as follows:

$$\mathbf{REAL, DIMENSION}(m, Np) :: x_e, y_e, z_e, x_i, y_i, z_i$$

$$\mathbf{REAL, DIMENSION}(m, Np) :: u_e, v_e, w_e, u_i, v_i, w_i$$

where the subscripts i and e, respectively, stand for ion and electron, the number m is the array size in each sub-domain. To ensure that the enough space are reserved to store the particle data due to the load-imbalance, m must be 10-30 % larger than the average number of particles. The number Np is the number of processors, and is the index used in the HPF "DISTRIBUTE" directive. As the particles move in time in the simulations, the physical position of some particles may cross the sub-domain boundaries, and move to the neighbor sub-domains. When a particle moves from one sub-domain to another, the data of the particle left the sub-domain must be sent to the appropriate neighbor processor every time step. Before updating and sending the particle data, we have to sort the particles that should be send to another sub-domain, and pack them in the working arrays: $CRi(:, Np)$, $CLi(:, Np)$, $CRe(:, Np)$, and $CLe(:, Np)$. The number of the ions and electrons sent in right and left are denoted by the arrays $ionspsR(Np)$, $ionspsL(Np)$, $lecspsR(Np)$, and $lecspsL(Np)$, respectively. In our HPF TRISTAN, we send both the packed arrays and their particles number arrays to the neighbor sub-domains as follows:

```
CRi=CSHIFT(CRi, -1,2)
ionspsR=CSHIFT(ionspsR, -1)
...
```

Figure 3 shows the example of the particle data distributions and communications. After both the particle numbers and the packed working arrays are sent and received by each appropriate processors, the received particles are sorted

and put into the appropriate part of the particle arrays in that sub-domain. The communications and sorting of these particles are performed in the particle manager [4].

5 Dual PentiumPro PC cluster

Our dual PentiumPro PC cluster consists of 16 PCs and each PC have dual $200MHz$ PentinumPros with 128MB EDD DIMM memories. The PCs in the PC cluster system are hooked through 100 Base-T ethernet with 100 Base-T switching Hub. Redhat Linux version 4.1 is used as their operating systems. The pgHPF compiler version 1.7 is installed for HPF computations.

6 Programming Comments on HPF Communications in PC Cluster

One of the most difficult HPF programming in our HPF TRISTAN code is the communication programming, especially, the determination of the buffer sizes which is used to pack the data to send to the neighbor processors. Of course, we can define a buffer size large enough to send the particle or grid data to neighbor processors at one time. However, as shown in Fig. 4, our experience shows that when the buffer sizes become larger than some critical values, in this case 1456 bytes in our PC cluster system, the communication suddenly becomes unstable, and the communication times suddenly jump up to 5 to 8 times larger than those less than the critical value 1456 bytes. As indicated in the figure, the communication times when the buffer size go beyond 1456 bytes are not uniquely determined and rather undeterministic. In order to avoid the sudden communication slow-down, we have to carefully to choose the buffer size. We have to split the particles or grids data into smaller pieces of buffers, pack the smaller data, and send the data to the neighbor processors one by one. Thus we can avoid the large slow-down of the simulations in this system due to the unstable HPF communications. In our HPF TRISTAN code, the buffer sizes can be varied and can be set without modifying the program. We can first evaluate the best buffer size and run the simulations. The best buffer size can be chosen as indicated in Fig. 4. For examples, in the figure, the best buffer sizes can be chosen between 640 to 1400 bytes.

The reason for performance degradation in communications with this longer packets than 1465 bytes are not yet investigated in detail. One possibility of this degradation is due to MTU of the ethernet. MTU is the Maximum Transmission Unit that IP is allowed to use for a particular interface. In general, the bigger the packet, the more data is transferred in the same number of packets, so the routers work better and you experience higher throughput. If your MTU is set too big however, your packets must be fragmented, or broken up, by a router along the path to the server. This results in a drastic decrease in throughput because the destination has to reassemble the packets that the routers took the

trouble to fragment along the way and this thread of the code in both server and router is usually sub-optimal. Of course, in our PC cluster the switching hub works as the router here.

7 HPF TRISTAN Code Results

Table 1. Time step=100, Particle Number =1200,000, Grid Number =185 × 65 × 65

Processors	total time×Proc. No. *(s)*	speed up S_p	efficiency $\varepsilon(\%)$	$\varepsilon_{eff-grid}(\%)$
1	4836	1.0	100.	100
2	7412	1.3	65.2	96.9
3	7249	2.0	66.7	95.4
4	7075	2.7	68.4	93.9
5	7287	3.3	66.3	92.5
6	7171	4.0	67.4	91.1
7	7236	4.7	66.8	89.8
8	7499	5.2	64.5	88.5
9	7937	5.5	60.9	87.2
10	7627	6.3	63.4	86.0
11	7843	6.8	61.7	84.9
12	7824	7.4	61.8	83.7
13	8019	7.8	60.3	82.6
14	8063	8.4	60.0	81.5
15	8588	8.4	58.3	80.4
16	8263	9.4	58.5	79.4
17	8453	9.7	57.2	78.4
18	8469	10.3	57.1	77.4
19	8617	10.7	56.1	76.4
20	8533	11.3	56.7	75.5

In Table 1, the parameter $\varepsilon_{eff-grid}$ is defined as:

$$\varepsilon_{eff-grid} = \frac{\text{total grid no. in decomposition direction - total guard cell number}}{\text{total grid no. in decomposition direction}}.$$

Table 1 shows the total times multiplied by the number of processors, speedups and parallel efficiency vs the number of processors. The total computation time of single processor are measured by the original version of TRISTAN code compiled by *pgf*77 compiler with the optimization level **-O2** option. Figure 5 shows the speed up vs processor number. Note that in the table we show the total CPU time multiplied by the processor number.

With fixing the problem size and increasing the processor number, the grid number in one sub-domain in decomposition direction is reduced gradually. For example, 60 extra ghost grid cells in total must be added to each sub-domains

in decomposition direction or in x for 20 processors. It is about 25 percents of the total grid number in decomposition direction in this case. Thus the communication overhead become to be insignificant comparing with the total PIC computation time as we increase the number of the processors. The increase of the communication overhead reduces the parallel efficiency in the table. If the communication overhead is insignificant, it is very hard to improve the parallel efficiency of the code without varying the problem size. However, even the most advanced parallel computer nowadays, it is not so easy to increase the problem sizes as we increase the the number of the processors due to the large data sizes we have to store in each simulation runs. Thus the optimal parallel efficiency of the scalable relation between the problem sizes and the number of processors are difficult to measure in our simulation. However, Fig. 5 shows the high linearity of our HPF TRISTAN code and the code scales well. In addition, with the HPF compiler overhead and the load-imbalance overhead due to the Earth dipole field located, the parallel efficiency around 60-65 % is affordable in this type of large scale simulations.

8 Concluding Remarks

In the present paper, we have successfully parallelized the three-dimensional full electromagnetic and full particle code using HPF. The code is originally the same as the TRISTAN code and the code is for the space plasma simulations. As shown in Fig. 5 and Table 1, fixing the problem size, our HPF TRISTAN code has a high linearity and scales well. However, our HPF code introduces about 70% overhead and the reason for this overhead is not yet investigated. We have also parallelized the three-dimensional skeleton-PIC code introduced by V. K. Decyk [4] in the same parallel algorithm [3] [4] using HPF. The HPF three-dimensional skeleton-PIC code introduces about 20% overheads[9]. One possibility to explain the larger overheads in our HPF TRISTAN code over the HPF skeleton-PIC code is that the more complicated data structures in HPF TRISTAN than those in the skeleton-PIC code. Our PCs in the cluster have no enough memory and this may degrade the performance of the PCs. Another possibility is the load-imbalance originated in the TRISTAN code as we discussed previously. Our TRISTAN code has the Earth dipole which simulates the Earth magnetospere only in one sub-domain, and this may cause a large load-imbalance. We would like to leave the detailed investigation to our future work.

The parallelization algorithm we used in our code is basically the same as [3][4]. We separate the communication parts from the computation parts. Thus the code can easily be converted to MPI or PVM code by replacing the HPF "CSHIFT" constructs to appropriate message passing interfaces. Our experiences show that the utilization of HPF "FORALL" or "DO INDEPENDENT" constructs in the data-parallel manner without separating the communication parts from the computation parts results in almost no gain of speedups or very poor speedups.

We have also compared the HPF skeleton-PIC code with the MPI or PVM skeleton-PIC code. The HPF code degradation of the total CPU time over the MPI or PVM code is only 10-15 % [9] in this case. Thus we expect that we should be able to enjoy the easier HPF programming with a very small performance degradation even in the more complicate codes like the TRISTAN codes.

Acknowledgment

The authors thanks Professor Viktor K. Decyk for his help using the skeleton PIC codes. The authors also thanks referees and Dr. Seo for invaluable comments regarding to the present paper.

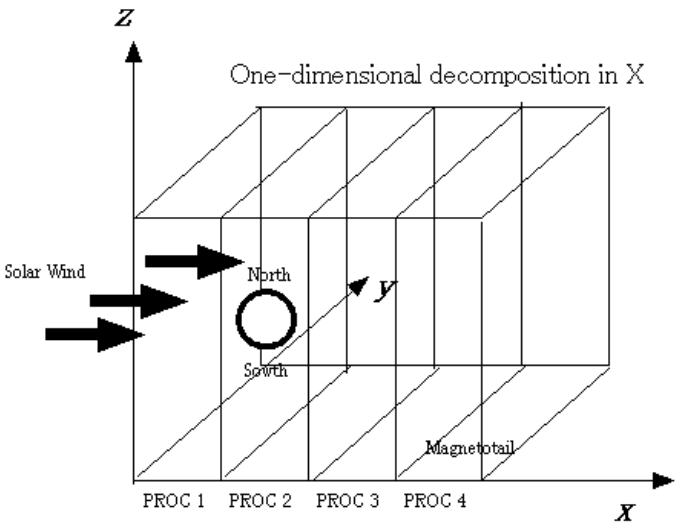

Fig. 1. Coordinate of the simulation domains and domain decomposition in x.

References

1. O. Buneman, TRISTAN. In: Matsumoto H., and Omura, Y. (eds.): Computer Space Plasma Physics: Simulation Techniques and Software. Terra Scientific, Tokyo, (1993) 67-84.
2. http://www.pgroup.com, (1998).
3. Liewer, P. C. and V. K. Decyk: A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes, *J. Comput. Phys. 85* , pp. 302-322 (1985).

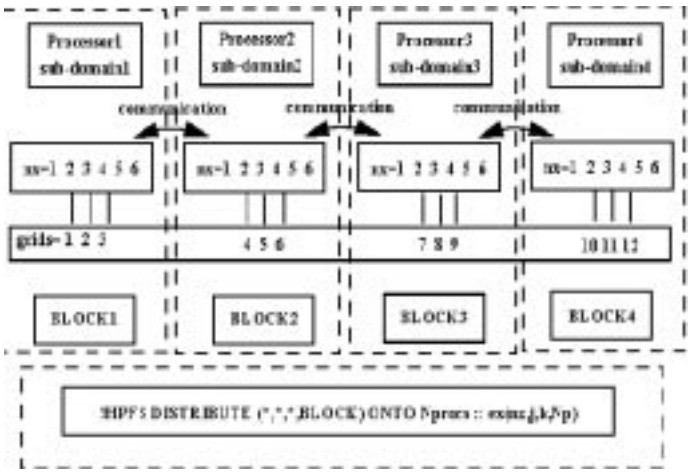

Fig. 2. Diagram of field array decompositions and communications, with processor number $Np = 4$, grid number in decomposition direction=12.

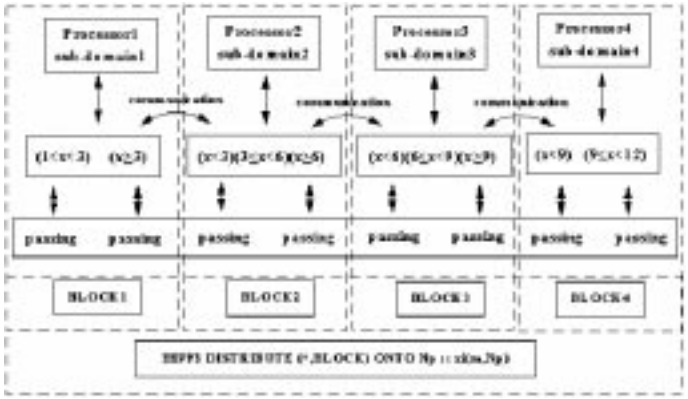

Fig. 3. Diagram of particle array decompositions and communications, with processor number $Np = 4$, grid number in decomposition direction=12.

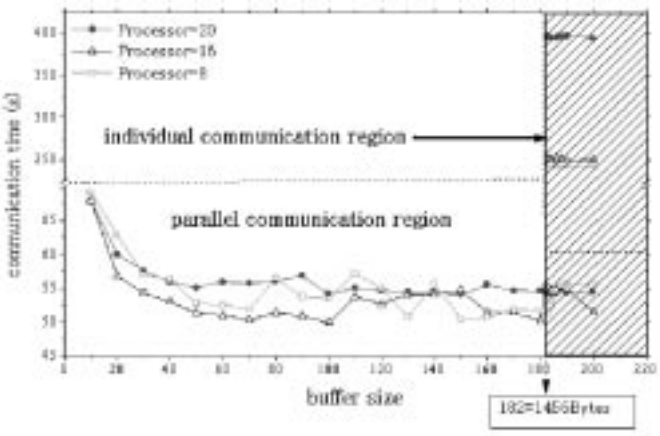

Fig. 4. Buffer sizes and CSHIFT communication times in HPF.

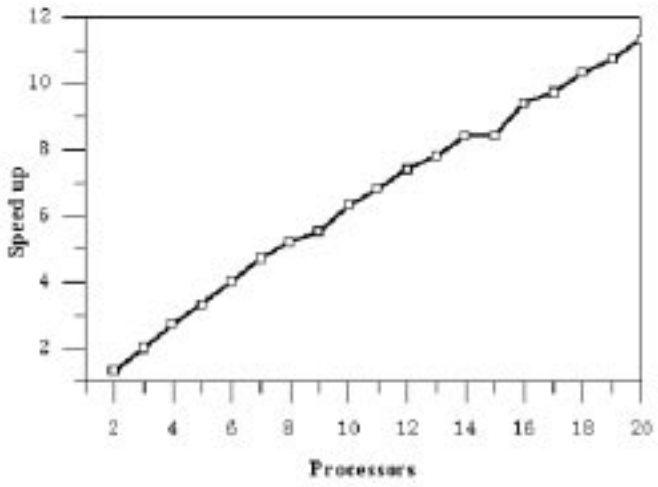

Fig. 5. Speed up vs processor number.

4. Decyk, V. K.: Skeleton PIC codes for parallel computers, *Comput. Physcs. Comm.* 87, Pp. 87-94, (1995).
5. C.K. Birdsall and A.B.Langdon, Plasma Physics via Computer Simulation, McGraw-Hill, New York, (1985).
6. D.W. Walker, Particle-in-cell Plasma Simulation codes on the Connection Machine, Computing Systems in Engineering, **1** (1991) 307-319.
7. C.H. Koelbel, et al., The High Performance Fortran Handbook, The MIT press,(1994).
8. Ian Foster, Designing and Building Parallel Programs, Addison-Wesley, (1995).
9. Cai, D., Q. M. Lu, and Y. T. Li, Scalability in Particle-in-Cell code using both PVM and OpenMP on PC Cluster, *Proceedings of 3rd Workshop on Advanced Parallel Processing Technologies* Pp. 69-73 (1999).